Single-picture reconstruction and rendering of trees for plausible vegetation synthesis

Anonymous

Abstract

State-of-the-art approaches for tree reconstruction either put limiting constraints on the input side (requiring multiple photographs, a scanned point cloud or intensive user input) or provide a representation only suitable for front views of the tree. In this paper we present a complete pipeline for synthesizing and rendering detailed trees from a single photograph with minimal user effort. Since the overall shape and appearance of each tree is recovered from a single photograph of the tree crown, artists can benefit from georeferenced images to populate landscapes with native tree species. A key element of our approach is a compact representation of dense tree crowns through a radial distance map. Our first contribution is an automatic algorithm for generating such representations from a single exemplar image of a tree. We create a rough estimate of the crown shape by solving a thin-plate energy minimization problem, and then add detail through a simplified shape-from-shading approach. The use of seamless texture synthesis results in an image-based representation that can be rendered from arbitrary view directions at different levels of detail. Distant trees benefit from an output-sensitive algorithm inspired on relief mapping. For close-up trees we use a billboard cloud where leaflets are distributed inside the crown shape through a space colonization algorithm. In both cases our representation ensures efficient preservation of the crown shape. Major benefits of our approach include: it recovers the overall shape from a single tree image, involves no tree modeling knowledge and minimal authoring effort, and the associated image-based representation is easy to compress and thus suitable for network streaming.

Keywords:
Tree reconstruction, Tree rendering, Vegetation synthesis

1. Introduction

Publicly available Digital Terrain Models (DTM) and associated aerial images have opened new possibilities for using real scenarios in video games and entertainment applications. Since vegetation is poorly represented in current DTMs, artists often populate the terrain with realistic tree models to support close-up views of the scenario (as required e.g. in racing video games). Unlike imaginary scenarios, where tree appearance can be left to artists' criteria, a substantial amount of modelling effort is required to create realistic tree models matching the local species of a given forest area (e.g. there are more than forty varieties of pine trees, each one native to specific regions around the world). This last requirement favors tree reconstruction over modeling, as the former better reproduces existing tree varieties and, in most cases, reduces authoring efforts.

In this paper we present a pipeline to reconstruct and render plausible trees. Unlike most existing approaches for tree reconstruction, which require either a point cloud or multiple photographs of each species, we extract the overall shape and appearance of the tree from a single photograph of the tree (Figure 1). This way we can benefit from georeferenced images from the web to synthesize trees for a particular location. We extract three basic pieces of information from the tree photograph. The overall shape is extracted from the crown silhouette via a bilaplacian filter. Crown color from the photo is used to synthesize a seamless, 360° color texture for the whole crown. Crown luminance is used to compute a relief texture through a shape-from-shading method, providing mid-frequency and high-frequency details on the crown shape.

Our method starts with a picture of a tree crown, together with a foreground/background segmentation of its foliage (Figure 1). From this picture, we generate two outputs: a *color map* representing the appearance of the foliage, and a *radial distance map* (RDM) storing the maximum distance from the crown center to the crown surface, i.e. for each unit vector we store the radius from the center to the outermost silhouette. Both textures are stored as cube maps and used at render time. The generation of the RDM requires creating a temporary mesh representing the overall shape of the crown, which is then combined with an estimation of the crown relief extracted from the color map using a shape-from-shading approach.

Once the RDM has been generated, the resulting crown volume can be used as input for a Space Colonization algorithm [1] to generate a branching structure at the desired detail level. Since we assume the crown is dense enough, we only need to reconstruct and display the main branches. This contrasts to previous approaches, which reconstruct a complete underlying branching structure beforehand and then add the leaves.

Our representation allows for efficient rendering of thousands of tree instances. Distant trees are rendered with an output-sensitive relief mapping approach. For close-up views, we switch to a representation using textured billboards clipped

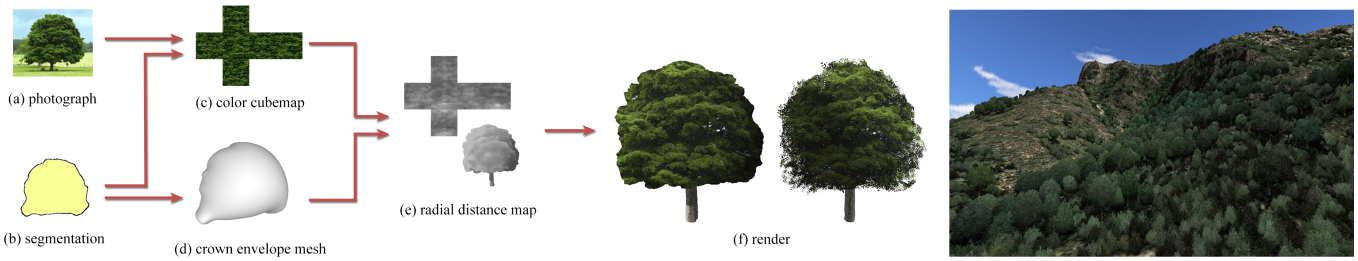

Figure 1: We present an algorithm to reconstruct trees from a single photograph. The resulting models can be used to synthesize plausible vegetation on top of arbitrary terrains. Our approach can benefit from georeferenced images from the web to populate landscapes with native tree species.

against the crown volume.

The rest of the paper is organized as follows. Section 2 reviews previous work on tree modeling, reconstruction and rendering. Section 3 describes the reconstruction of the tree, and Section 4 is devoted to tree rendering. Section 5 discusses our results with a variety of tree species. Concluding remarks are provided in Section 6.

2. Previous work

Tree modeling The lack of the hardware needed to capture and process real tree models directed early efforts towards tree modeling. The first contributions focused on the effect of certain parameters (such as branching angle and branch length) on tree shapes [2]. These findings were applied to recursive algorithms which produced the first tree structures resembling their real counterparts [3]. The parameters of these algorithms allowed the user to alter the generated model in real-time, making it possible to examine the model space interactively [4]. One of the main drawbacks of such techniques is the large number of parameters required to specify a particular model [5]. It is possible though to obtain values for these parameters from a given tree model that can then be used to generate similar ones [6]. All these methods allow generating the branching skeleton of a tree but they transform the resulting hierarchy into a 3D model using simple modeling algorithms. Bloomenthal et al. [7] examined the transformation process proposing techniques to represent the trunk, branches, and bark of a tree more faithfully.

Another set of techniques were introduced by Lindenmayer [8] that, exploiting the capabilities of formal languages, managed to imitate plant development. L-systems have been widely used for modeling all types of plants and have been extended to support most of its peculiarities. These extensions include the integration of production rules to represent the development of the plant over time [9], the interaction with the environment [10], the expression of plant attributes based on their spatial location [11], and many others [12]. Despite the difficulty of predicting the outcome produced by this type of grammars, it is possible to control the process and produce the desired output [13]. Lintermann et al. [14] introduced a set of algorithms that encapsulated components and structures associated with key parts present in most plants. These components could be combined using software similar to most geometric modelers, thus resulting in a more intuitive way of modeling plants.

One recent strategy is based on the observation of the factors that influence the final shape of a tree. In particular, competition for resources (sunlight, space) by different branches of a tree seems to be critical for the general shape of temperate-climate trees. The space colonization algorithm [1] uses this fact to establish a set of attractors in the volume defined by the tree's foliage, which is iteratively conquered by the branches as they occupy the available space. Palubicki et al. [15] extended this algorithm using a signaling mechanism to mimic different types of growth. It is also possible to provide the overall crown shape using a gesture based system that guides the resulting tree's growth [16], reducing modeling time while maintaining the artist's ability to obtain the desired result.

Other factors such as wind and surrounding space also influence a tree's growth. Pirk et al. [17] presented a technique that made it possible for content creators to change a tree's position inside a scene and see its shape adapt to changes in light distribution and the occupation of surrounding space. In [18] it was extended to include the effect of wind.

Tree reconstruction Only recently interest has shifted to the reconstruction of trees, either from multiple photographs or point clouds obtained from laser scanners. Shlyakhter et al. [19] proposed an algorithm that took several photographs as input and used the segmentation of the crown to construct the tree's visual hull. The resulting shape was filled with a plausible tree skeleton that they augmented using L-systems. To improve the segmentation stage Reche et al. [20] applied an image matting algorithm, combining the results to generate an opacity volume. The reconstructed trees were rendered view-dependently attaching billboards to opaque cells. Another possible strategy would be using the volume to generate the initial positions of a 3D flow simulation, where the trunk and first-order branches were used as attractors [21].

Some techniques take a hybrid approach. In order to generate the model, they transform their input image collection into a point cloud using a structure-from-motion algorithm. Quan et al. [22] used the generated point cloud to extract and segment individual leaves allowing the user to model the branches. On the other hand, Tan et al. [23] synthesized hidden branches from visible ones with texture-based synthesis algorithms and then added the crown instantiating leaf clusters identified in the point cloud. Also starting from a point cloud (in addition to an

example of a mesh leaf) the technique by Bradley et al. [24] is capable of generating plausible foliage configurations. In order to do this, it fits the exemplar leaf to the point cloud, extracting a statistical model of the shape, appearance, and transformations between neighbors.

Scanned point sets have also been used to extract tree geometry. Xu et al. [25] classified points into two types: leaves and branches. Their method builds a graph connecting neighbors that is used to produce a skeleton, extends it into the tree crown and transforms it into a mesh. Other approaches refine the initial branch-structure graph according to some optimization criteria [26]. It is also possible to decompose a tree's crown into geometry structures called lobe-textures [27].

A 3D tree model may also be generated from a 2D sketch assuming that trees spread their branches far apart from each other [28]. Such system also allows users to apply gesture-based editing operations and manually generate trees from given examples. Wither et al. [29] made use of successive silhouettes sketched at different zoom levels to create a 3D tree.

Most reconstruction methods that take photographs as input need either several images to work properly or significant user interaction. The technique proposed in [30] is a notable exception and our direct competitor, at it is able to extract the branching structure from a single image of the tree. The user draws one stroke in the photograph to identify the crown, and another one (or sometimes more) for the branches. The crown is segmented and the visible branches are converted to 3D using the approach proposed in [28]. The initial skeleton is extended into the crown by iteratively substituting an existing branch by a subtree from a database. Unlike [30], our image-based representation is much more compact and thus suitable for efficient streaming and rendering of forest scenes. Our approach also supports arbitrary camera rotations around the tree, whereas [30] is intended for still renderings of front and side views of the tree due to the direct use of the input image as texture. On the other hand, we focus on the reconstruction of tree foliage instead of the branching structure that, for dense trees, would be hardly visible. As a downside, we cannot reconstruct sparse trees, which are correctly handled by [30].

Tree rendering Plants and ecosystems are extremely complex objects and their interactive rendering is still a challenging problem. Research focused on rendering large forests has developed multi-scale systems [31] that use several representations of each tree. One option is to use particle systems [32] as they can represent irregular 3D volumetric structures with ease. One could also substitute a plant model by a set of points and line primitives [33]. Such a model allows to smoothly reduce detail by rendering a progressively smaller subset of those basic primitives. Then, dividing the scene hierarchically, we can apply different ratios of reduction to each region. Gilet et al. [34] proposed a technique that uses point clouds to represent vegetation. Several instances of one or more blocks of vegetation compose a landscape and each block is rendered using either its original polygonal model or a decimated point cloud generated from it. Decaudin and Neyret [35] covered the ground of the forest they wanted to display with a volumetric texture that was rendered using texture slicing. Andujar et al. [36] synthesize procedural vegetation visually compatible with low-resolucion aerial images by using a rotationally-symmetric crown model which is rendered by modulating the aerial color with prototype leaf textures. Compared to [36], our approach automatically generates a faithful representation of the crown volume and appearance from a photograph of the tree crown, resulting in clearly distinguishable tree specimens.

More recently this problem has been approached from a simplification viewpoint, the difficulty being that vegetation is composed of a large number of already very simple elements. Cook et al. [37] addressed it by rendering them using a randomly selected subset of elements statistically altered to preserve the overall appearance. Then Neubert et al. [38] improved upon the previous technique by defining a priority based pruning that preserved the model's silhouette.

The massive amount of geometry present in a forest scene is not the only difficulty. Realistic lighting is also a challenge at all levels of detail. Behrendt et al. [39] approximated plant models using billboards generated from a clustering algorithm applied to the model's components. Their technique incorporated realistic illumination via spherical harmonics, while approximating distant vegetation with shell textures. Bruneton and Neyret [40] presented a system that uses two scales based on light fields and height fields to render millions of trees in real-time with realistic lighting.

3. Crown reconstruction

3.1. Input photograph assumptions

Our method starts from a single picture of a tree and a segmentation mask for its crown foliage area. Analyzing automatic matting algorithms was out of the scope of this work. We interactively generate this segmentation in a few seconds using the segmentation tools described in [41], and applying a contraction and smoothing afterwards to the segmented contour.

Currently, our approach works best when the following conditions about the input photograph are met:

- The crown foliage should be dense, the underlying branching structure should be hidden or hardly visible.
- The crown brightness should be influenced by self-occlusions (this is often a result of dense crown foliage).
- The crown shape can be approximated by a single volume. In the future we might extend the method to deal with trees with sparse foliage nuclei.
- The leaf system should not exhibit strong, large-scale directional patterns as in, e.g. palm trees, which are not supported by our approach. Small scale directional patterns (e.g. preferred leaf orientations due to phototropism and high-frequency features alike) have little impact on our synthesized textures and are thus well supported.
- If the crown is not completely visible, the artist can specify a segmentation to be used for the crown shape, together with another segmentation, to be used for texture synthesis, identifying pixels not belonging to the crown.

Figure 2: (Left) Individual of Fagus Sylvatica. (Center) Inside a fagus sylvatica forest; the branching structure of close trees is clearly visible. (Right) The same forest seen from about 100 m above, now with a much more dense and uniform appearance.

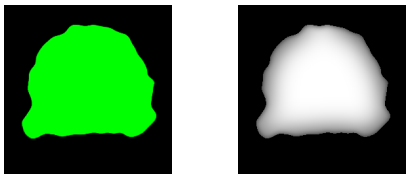

Figure 3: From the tree segmentation (left) we can compute a heightmap that represents the front half of the envelope mesh (right) using the bilaplacian filter.

Figure 2 shows one example of tree species (*fagus sylvatica*) that does not follow our assumptions. It grows a dense crown but its branches are very separated and with a sparse foliage, thus revealing the underlying branching structure. Even for these cases, their global appearance as seen from afar is dense and uniform, and we can still obtain a plausible reconstruction suitable for mid-range and distant views.

3.2. Envelope mesh

The first step of the reconstruction consists in converting the silhouette given by the segmentation of the input image into a rough approximation of the tree crown. We will refer to this volume as the envelope mesh. Sketch-based systems [42] provide algorithms to *inflate* a mesh from a simple boundary, but results with typical tree silhouettes are rather poor (see side views in Figure 8) as elevation values on spine vertices directly depend on the distance to the silhouette (to make wide areas fat, and narrow areas thin).

Since we cannot assume a high-quality segmentation of the crown's silhouette, we decided to generate a relatively smooth envelope mesh by generating a heightmap over the segmented image (see Figure 3). This heightmap will provide the frontal half of the envelope mesh, and its inverted version will provide the other half. Due to the ill-defined nature of this problem, we fix the silhouette points of the heightmap on the $z = 0$ plane (z represents height) and we ask their tangents to be parallel to the z axis, while requiring that all points inside the crown reduce their bending energy. We achieve this by minimizing the thin-plate energy defined by a biharmonic equation inside the crown C, while restricting the silhouette S to have $z = 0$. Written as a continuous optimization problem, we want to solve

$$\min_z \int_C \nabla^4 z, \quad \text{such that} \begin{cases} z \geq 0 & \forall p \in C \backslash S \\ z = 0 & \forall p \in S \\ z' = f(x,y) & \forall p \in S \end{cases}$$

where ∇^4 is the biharmonic operator and f is a positive real function controlling the gradient of the distance field at the silhouette points.

The optimization problem above can be written as a linear system for the z values using the bilaplacian filter [43]. We discretize the domain C using our input segmentation as a 2D grid discretization. In this discretized domain, tangets are specified as follows. We define the outer silhouette S_o of the image as the set of pixels outside C but adjacent to a silhouette pixel. In this setting, for each inner pixel (i, j):

$$\begin{array}{ll} z(i,j) \geq 0 & \text{if } (i,j) \in C \backslash S \\ z(i,j) = 0 & \text{if } (i,j) \in S \\ z(i,j) = -f(i,j) & \text{if } (i,j) \in S_o \end{array}$$

Then, for each unknown z we create the equation obtained from the convolution of its corresponding point with its neighbors, representing the thin-plate energy computation:

$$\sum_{\substack{-2 \leq d_i \leq 2 \\ -2 \leq d_j \leq 2}} M_{d_i+2, d_j+2} \cdot z(i+d_i, j+d_j) = 0 \qquad (1)$$

$$M = \frac{1}{16} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & -8 & 2 & 0 \\ 1 & -8 & 20 & -8 & 1 \\ 0 & 2 & -8 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

This convolution is the result of the application of two successive laplacian filters (a well known edge detector image filter) as the bilaplacian itself is the result of computing the laplacian of a laplacian.

Solving the system yields promising results, but there are still some issues (as shown in Figure 4-left). Due to the way we apply the tangents' constraints using the outer silhouette on the grid, they cannot be defined as we wanted —as vectors pointing in the z direction— because there will always be a small displacement in x and y between the outer silhouette and the real one. Therefore, the resulting envelope will never look as smooth as we want. Increasing the negative displacement for pixels in S_o alleviates this problem but tangents will never match with the mirrored ones at the silhouette.

In order to properly express tangent constraints we decided to extend the previous linear system to 3D with unknowns for x and y as well. Thus, inner vertices ($\in C \backslash S$) will no longer be restricted to a grid arrangement. In this new setup, grid vertices have coordinates

$$\begin{array}{ll} (x_x, x_y, x_z) & \text{if } (i,j) \in C \backslash S \\ (i,j,0) & \text{if } (i,j) \in S \\ (i', j', f(i', j')) & \text{if } (i,j) \in S_o \end{array}$$

where x_x, x_y and x_z are the unknowns, and (i', j') is just the projection of $(i, j) \in S_o$ onto the closest silhouette pixel. We

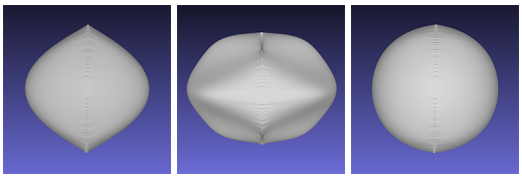

Figure 4: Inflating the envelope while keeping the x and y components fixed does not produce the desired results (left). However, expressing the minimization problem in 3D helps solve this problem (right) but only if a 8-connected silhouette is used. A 4-connected silhouette produces unacceptable artifacts (center).

use Equation 1 independently on each unknown to solve the problem.

Pixel connectivity plays a big role in the reconstruction process. The set of pixels in S contains pixels classified as inside the crown but adjacent to pixels outside. If we assume 4-connectivity for the silhouette pixels, results are not sufficiently smooth (e.g. we cannot reconstruct a sphere from a circle boundary, as shown in Figure 4). Using 8-connectivity, however, solves these artifacts.

The first columns of Figure 8 show some tree silhouettes and the resulting crowns based on the envelope meshes. As for function $f(x,y)$, artists could chose either a constant function $f(x,y) = k$ (for roughly spherical crowns) or a quadratic function with the form $f(x,y) = ay^2 + 1$ (for approximately conical trees). In both cases we assume Y is the vertical axis pointing downwards, and (x,y) are normalized pixel coordinates. Next sections explain how we add color and relief details to these meshes.

3.3. Color synthesis

A single picture of a tree contains less than a half of the tree's outer surface. However, it is reasonable to assume that the occluded parts will have a similar appearance than those visible. We thus use a texture synthesis approach to recover missing parts.

First, the segmented photograph is scaled to have unit aspect ratio, and placed at the center of a 4:3 canvas such that it occupies 5/12 of its width and 5/9 of its height. We do this under the assumption that the original photograph must cover a face of the output cubemap plus about a third of the neighboring faces. We then fill the rest of the canvas using the texture synthesis algorithm from [44].

The color cubemap is constructed by cropping the six faces of this synthetic texture, as shown in Figure 5, and applying a local synthesis again on the bands around the borders of the faces that were not originally in contact. This additional step guarantees continuity across all cubemap edges.

3.4. Relief synthesis

Our relief estimation method is inspired by shape-from-shading (SFS) algorithms like [45]. Unfortunately, we can assume very little about the lighting conditions of the input photographs, and typical assumptions in SFS approaches (known reflectance model, known direction of light sources) do not

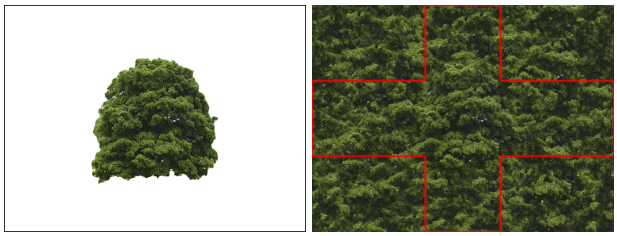

Figure 5: Left: placement of the segmented photograph inside the canvas to be synthesized. Right: synthesis result and cubemap outline. Notice that the synthesized result includes the segmented photograph.

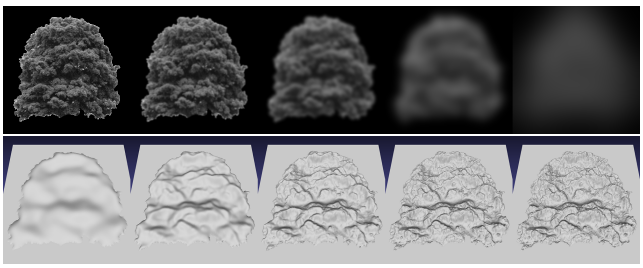

Figure 6: Gaussian decomposition pyramid (top) and multiscale relief generation (bottom). Each heightfield is the result of adding its corresponding Gaussian pyramid level to the previous one, and decreasing the weight at each step.

hold in our case. Furthermore, tree leaves form complex structures with high-frequency variation across the image, which is a worst-case scenario for SFS approaches, which describe the surface shape either in terms of the surface normal, or in terms of the heightfield gradient.

Since we assume the envelope mesh already captures the rough shape of the crown, we directly estimate depth from luminance. Despite the fact that a tree has a huge number of self-occlusions from the different branches and leaves, a fraction of incoming light is likely to reach occluded portions due to light scattering and non-opaque leaves. The deeper we are inside the crown, the darker we expect it to be. Again we assume a dense input crown, preventing deep areas close to the crown center from receiving direct light.

We start by computing the luminance of the input image and normalizing it to be in the $[0,1]$ range. Then, we compute a blur pyramid of this luminance image, similar to the *Gaussian pyramid* of Burt and Adelson [46]. We also tried a Laplacian decomposition, but the relief obtained from the Gaussian is more prominent due to lower frequencies being counted multiple times.

Once we have N levels of the pyramid (we tested $N = 4$ and $N = 5$), we combine them using an exponentially-decreasing set of weights. Lower levels (blurrier) will have larger weights because they represent large-scale (and potentially deeper) structures, while higher levels are usually noisier and just add small fine details to the relief.

Figure 6 shows an example of a relief reconstruction; the figure uses the input tree image for clarity; we extract the relief from the synthetic cube map described above.

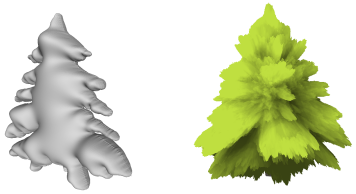

Figure 7: Base envelope mesh computed from a segmentation (left). Tree crown created from three rotated copies of the envelope and relief perturbation (right).

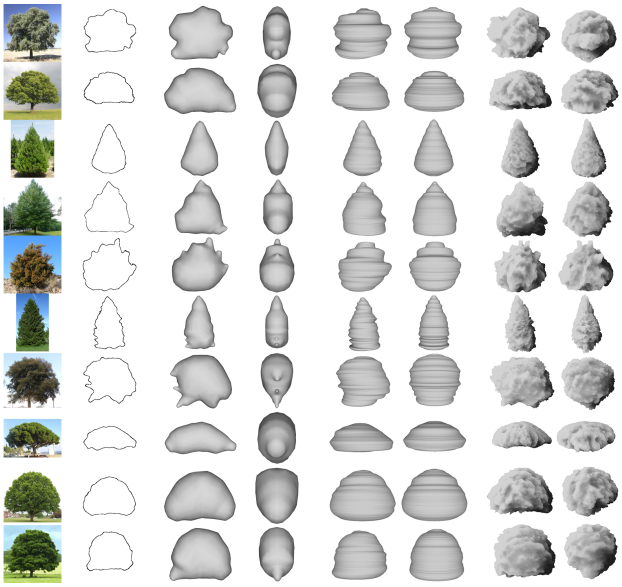

Figure 8: From left to right: input photograph, segmented silhouette, front and side views from a sketch-based system [42], front and side views using a function-based crown model from [36], and front and side views of our reconstructed crowns.

3.5. Radial distance map

The final representation of the crown shape is the radial distance map (RDM). For each direction, we store the distance from the crown center to the crown surface. This representation is the result of perturbing the crown envelope with the relief values extracted from the color cubemap. At build time, we just render the envelope mesh and compute the final radius on a fragment shader as $\|p-c\| \cdot (1+\rho \cdot (h(p-c)-0.5))$, where p is a point on the envelope surface, c is the center of the crown, $h(p-c)$ is the relief cubemap value in the direction $p-c$, and ρ is a parameter that controls the weight of the relief map. In order to improve the overall shape of certain types of trees, the artist may choose to add several rotated copies of the crown envelope (as an example see Figure 7). This just involves rendering the envelope mesh multiple times and keeping the maximum radius value (through depth test) in the output cubemap.

Figures 8 and 16 show some reconstructed volumes.

3.6. Branches and leaves

In order to support highly detailed models, a branching structure for the crown can be generated with the Space Colonization algorithm [1] operating on the reconstructed RDM. Other resource competition algorithms like shadow propagation [15] could be used for this purpose, but the Space Colonization is sufficient for our needs and easy to implement. Given a RDM R, we first generate a set S of N attraction points using a blue noise distribution. Points outside the crown envelope represented by R (checked by a simple cubemap lookup) are discarded. We then run the Space Colonization to generate the branching structure. Figure 9 shows an example crown and the branching structure obtained. Note that only the main branches are represented.

Figure 9: Base envelope mesh computed from a segmentation (left). Textured crown obtained with relief perturbation (center). Branching structure (right).

Figure 10: Input leaflet (left) and automatically generated billboards using Space Colonization from a conical point cloud (center) and a spherical point cloud (right).

In order to keep real time framerates during rendering, we use billboards to represent groups of leaves and small branches. These billboards are created by instantiating multiple copies of a leaflet (a photograph of leaves or small branches, or even a 3D mesh if available). The distribution and arrangement of leaflets inside each billboard is also based on the Space Colonization algorithm with an arbitrary input volume. We found that using simple shapes like spheres or cones yields convincing results (see Figure 10). Conical volumes yield billboards representing branchlets that will be connected from the base of the cone to the branching structure. Spherical clouds, although less realistic when seen in isolation, can be rotated arbitrarily around the connecting point - their center - and thus can be oriented at runtime to face the viewer at any time.

Fully-detailed tree models can also be generated by increasing the refinement of the branching structure. Figure 11 shows a couple of tree models generated using this approach in about 90 seconds (Python code running on Blender). Notice that the final shape of the tree models closely matches that of the crown envelopes. In the examples, we used (in [1] notation) $N = 10,000$ attraction points (to ensure a dense distribution of points within the crown), a *kill distance* $d_k = 0.01r$ (r being the average radius of the crown envelope) to ensure that the branching system provides a dense coverage of the crown volume, and an *intern-*

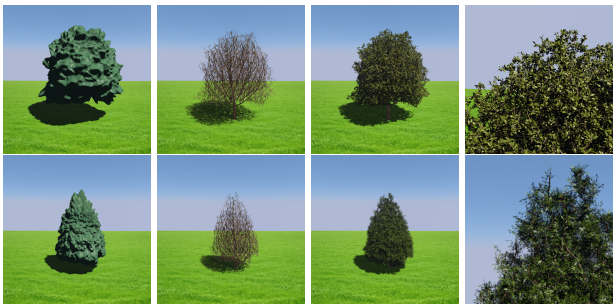

Figure 11: Fully-detailed models created by generating attraction points within the crown envelope and then using a space colonization algorithm. The images show the crown envelope, the branching system, and the final tree (full/detail).

ode length $D = 0.5k_d$. The resulting models have around 300 K textured quads (leaves) plus about 100 K quads (branching system) and their use should be reserved to very close-up views.

4. Crown rendering

The algorithms above construct a radial distance map and a color map (both represented as cube maps) for each tree photograph given as input. This representation can be rendered with increasing levels of detail, as described below.

4.1. Direct rendering

Our first rendering method implements a direct visualization of the RDM through fragment-based relief mapping. Relief mapping approaches [47] cannot compete with current tessellation engines when close-up views of detailed geometry are required, but offer excellent performance for distant objects (especially in deferred shading setups) since their rendering cost is output-sensitive, i.e. the cost directly depends on the number of covered pixels [48]. This is thus the LoD we use for rendering distant trees.

The classic approach for relief mapping [47] is to compute the ray-heightfield intersection by sampling points along the ray, first using linear search to find a sample inside the object, and then refining the intersection point through binary search. For each sampled point, the height of the sample is compared with the depth stored in the relief map to determine whether the sample is inside or outside the heightfield. We apply this idea to radial distance maps. When traversing the ray along the viewing direction, ray samples are computed in world space (instead of tangent space). For a ray sample P, we compute $\vec{v} = (P - C)/R_{\max}$ (C is the crown center and $R_{\max}$ is the maximum crown radius), and check if P is inside the crown by comparing $\|\vec{v}\|$ with the radial distance value stored for direction $\vec{v}$.

We considered two options for computing the fragment color. The simplest option is to get the color directly from the color cube map. However, crown parts along the same radial direction get the same color from the cube map, resulting in some texture stretching that might be noticeable in trees with prominent branches and abrupt relief changes. Alternatively, we can get the fragment color from the central part of the synthetic 2D texture (Figure 5), using the same texture coordinates we would use for a planar billboard. This second option provides higher quality images, but the re-usage of the same texture portion for all view directions becomes apparent when rotating the camera around a closep-up tree. We thus use this second option only for distant trees. Notice that with both options the rendered tree silhouette matches that of the RDM.

Figure 12: Rendering clipped billboard clouds: (a) leaf texture; (b) segmentation showing individual leaf centroids, (c) per-fragment clipping, (d) per-leaf clipping, based on the inside/outside classification of the leaf centroid.

4.2. Clipped billboard clouds

For closer views we add high-frequency details by drawing a billboard cloud with an RGBA texture showing leaves and small branches. Billboard centers are placed on points of the branching structure generated using the Space Colonization algorithm (see Section 3.6). A single vertex buffer is shared among all tree instances of the same species. The actual number of points to render (converted into quads in the geometry shader) varies for each instance, depending on the distance to the viewer, as in [37].

In order to preserve the overall crown shape of the species, we clip the geometry outside the crown volume. Again, a point-inside-crown test requires a single texture lookup to the RDM. This clipping operation can be performed at different granularity levels. Discarding complete billboards based on their center (e.g. in the geometry shader) is the simplest option, but tends to produce popping artifacts around the tree silhouette when large textures are used. Discarding individual fragments faithfully preserves the crown shape (except for transparent texels in the leaf texture), but tends to cut individual leaves in the texture. A better option is to discard fragments based on the centroid of the individual leaves in the texture (Figure 12). This requires additional segmentation information where individual leaves point towards the cell centroid. We automatically generate this information when we create the billboards. This technique yields better silhouettes at the only expense of an extra texture lookup.

Apart from the leaf centroid, we also store other information in the billboards (see Figure 13). Normals and precomputed ambient occlusion are used during shading. Depth is used to reduce the artifacts when two billboards cross. The type bit (shown in black and gray for clarity) separates the leaves from the branches, and we use this information during specular lighting. Finally, the texture coordinates refer to the coordinates used to instantiate the original photograph (see Figure 10).

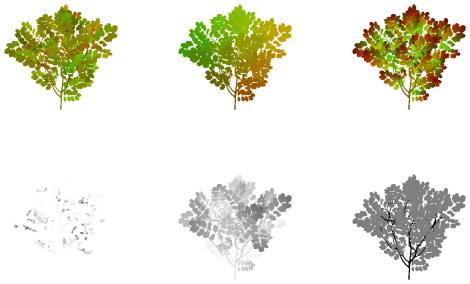

Figure 13: Information stored for the generated leaves billboards besides the RGBA components. Left to right and top to bottom: normals, leaf centroids, texture coordinates, ambient occlusion, depth, type bit.

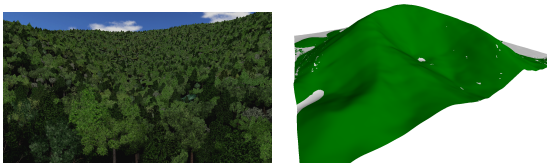

Figure 14: Test scene used for performance analysis. The image on the right shows tree areas.

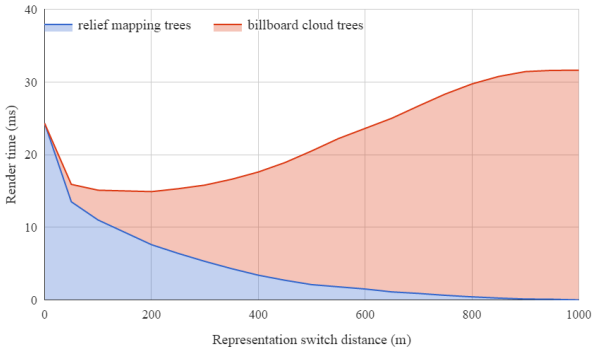

Figure 15: Accumulated rendering times of both LoD representations depending on the switch distance.

5. Results

5.1. Rendering performance

We measured the rendering performance of our tree representations on a dense forest scene (Figure 14). Accurate selection and placement of appropriate trees for a given terrain is out of the scope of this paper (see e.g. [49] for segmentation of aerial images, and [36] for modulating the appearance of tree instances with color from aerial images). We thus randomly placed 50K trees in green areas of the aerial image, following a blue noise distribution. Tree instances had varying sizes. We chose a view with a large number of visible trees (42K trees, Figure 14-left) with a broad range of distances (up to about 1,000 m).

Rendering times were measured with different distance thresholds (d) for the LoD switch. Figure 15 shows average rendering times on NVIDIA GTX 970 hardware on a high-quality setting: Full HD (1920×1080) resolution and 8× multisampling. All tree instances within d distance were rendered using the detailed (billboard cloud) representation, using a varying number of billboards per instance (from 16 to 144) according to the screen-projected area. Tree instances beyond d were rendered using the direct (relief-mapping) representation.

For the detailed representation, the throughput varied from 700 trees/ms (only the closest trees being rendered with the detailed representation) to 1,300 trees/ms (all trees rendered as billboard clouds). For the direct rendering, the throughput varied from 1,700 trees/ms (all trees rendered through relief mapping) to 12,000 trees/ms (only distant trees rendered with relief mapping). For distant trees, the relief mapping was up to one order of magnitude faster than billboard clouds, mainly due to its output sensitive nature.

From a visual quality point of view, we found out that the distance threshold d should be greater that 150 m (for smaller values the artifacts of the relief representation become too apparent). From a performance point of view, optimal d values were about 200 m (due to the relief map representation to become inefficient for tree instances covering a large part of the viewport, whereas the per-instance cost for the detailed representation remains roughly constant once they reach the minimum number of billboards). Frame rates for $d = 200$ m were above 60 fps even for this dense forest area.

5.2. Memory footprint

For each tree species we need to store the following data: a RDM encoded as a single-channel cubemap ($6 \times 256 \times 256$), a RGB color texture (1024×768), a small number of billboard textures (512×512, encoding RGBA channels plus 8 bytes per pixel for the additional information), and a branching structure (about 200-400 3D points). Note that with only the RDM and the color texture (4.5 MB of uncompressed data) we can already render trees with the direct representations. For the detailed representation we used four billboard textures per species, each texture taking 3 MB of uncompressed data. The total memory footprint per species is less than 17 MB. Using standard compressed texture formats, the footprint per species is about 5 MB, which is negligible when compared with aerial imagery.

5.3. Tree reconstruction

The results of our crown reconstruction algorithm with various tree photographs are shown in Figure 16. Notice that the overall crown shape and appearance are preserved. The average construction time (on an Intel Core i7 CPU) including manual intervention was 5 min. User segmentation of the crown shape in the photograph took less than a minute in an interactive editor. The most expensive step (about 1-2 min) was the synthesis of the color texture (Section 3.3). In our tests, we generated a 1024×768 image. The rest of the reconstruction steps, including solving the bilaplacian linear system, the computation of the Gaussian pyramid and the construction of the RDM took less than 5 s. Space Colonization to produce the main branching structure and each billboard run in less than a second. Since we

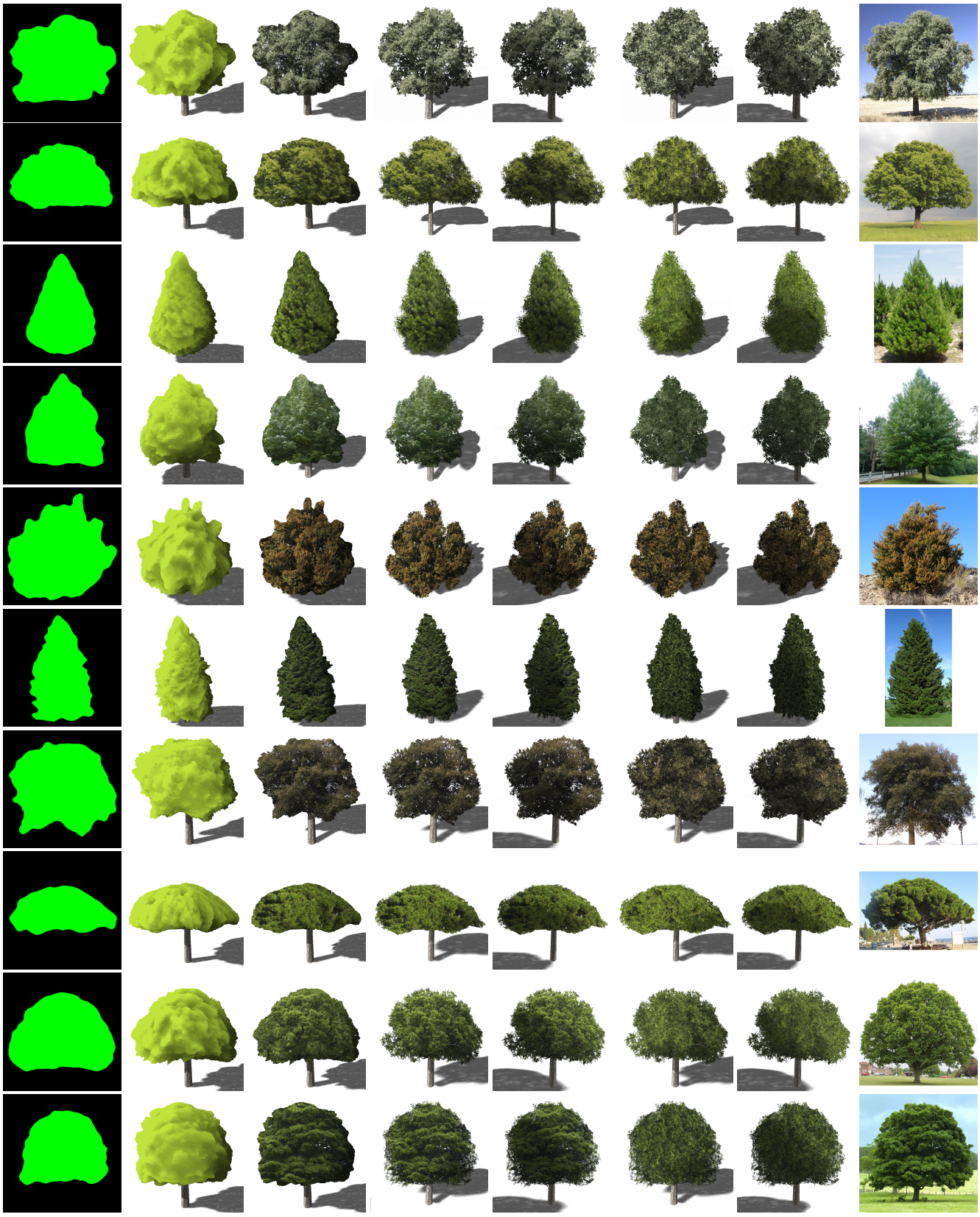

Figure 16: Examples of trees reconstructed using our method. From left to right: segmentation mask, front view of the volume represented by the radial distance map, the same view using the color texture, two lightings of the billboard cloud textured from the color texture, two lightings of the billboard cloud using the color from the billboards, and the original photograph.

only wanted the overall structure, we used only a few thousand attractors.

6. Conclusions

In this paper we have proposed a pipeline for helping artists to create plausible tree models from individual photographs with minimal authoring efforts. The resulting models can be used to synthesize local vegetation on top of DTMs representing real scenarios. The fact that a single tree photograph suffices allows artists to benefit from available georeferenced photographs to populate arbitrary forest areas. Unlike competing approaches, reconstructed trees are encoded in a compact, image-based, easy to compress representation, suitable for network streaming, and support efficient rending from arbitrary view directions.

Some assumptions (Section 3) are needed for our approach to work. Essentially, our method works with trees with dense, single-nucleus foliage without large-scale directional features.

Interesting avenues for future work include the support of sparse crowns and multiple-nuclei foliage, the computation and use of opacity values from the input image, and automatic extraction of leaf patches for the billboard textures. Our approach would greatly benefit from advances in automatic segmentation of tree crowns in photographs, as well as improvements in automatic recognition of vegetation in aerial images.

References

[1] A. Runions, B. Lane, P. Prusinkiewicz, Modeling Trees with a Space Colonization Algorithm, in: Proceedings of the Third Eurographics Conference on Natural Phenomena, NPH'07, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2007, pp. 63–70.

[2] H. Honda, Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body, Journal of Theoretical Biology 31 (2) (1971) 331 – 338.

[3] M. Aono, T. Kunii, Botanical tree image generation, IEEE Comput. Graph. Appl. 4 (5) (1984) 10–34.

[4] P. E. Oppenheimer, Real time design and animation of fractal plants and trees, in: Proc. of SIGGRAPH '86, ACM, 1986, pp. 55–64.

[5] J. Weber, J. Penn, Creation and rendering of realistic trees, in: Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95, ACM, New York, NY, USA, 1995, pp. 119–128.

[6] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, B. Benes, Inverse procedural modeling of trees, Computer Graphics Forum 33 (6) (2014) 118–131.

[7] J. Bloomenthal, Modeling the mighty maple, SIGGRAPH Comput. Graph. 19 (3) (1985) 305–311.

[8] A. Lindenmayer, Mathematical models for cellular interactions in development: Parts i and ii, Journal of theoretical biology 18 (3) (1968) 280–315.

[9] P. Prusinkiewicz, M. S. Hammel, E. Mjolsness, Animation of plant development, in: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '93, ACM, New York, NY, USA, 1993, pp. 351–360.

[10] R. Měch, P. Prusinkiewicz, Visual models of plants interacting with their environment, in: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96, ACM, New York, NY, USA, 1996, pp. 397–410.

[11] P. Prusinkiewicz, L. Mündermann, R. Karwowski, B. Lane, The use of positional information in the modeling of plants, in: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, ACM, New York, NY, USA, 2001, pp. 289–300.

[12] P. Prusinkiewicz, A. Lindenmayer, The Algorithmic Beauty of Plants, Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[13] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, V. Koltun, Metropolis procedural modeling, ACM Trans. Graph. 30 (2) (2011) 11:1–11:14.

[14] B. Lintermann, O. Deussen, Interactive modeling of plants, IEEE Comput. Graph. Appl. 19 (1) (1999) 56–65.

[15] W. Palubicki, K. Horel, S. Longay, A. Runions, B. Lane, R. Měch, P. Prusinkiewicz, Self-organizing tree models for image synthesis, in: ACM SIGGRAPH 2009 Papers, SIGGRAPH '09, ACM, New York, NY, USA, 2009, pp. 58:1–58:10.

[16] S. Longay, A. Runions, F. Boudon, P. Prusinkiewicz, TreeSketch: Interactive Procedural Modeling of Trees on a Tablet, in: K. Singh, L. B. Kara (Eds.), Eurographics Workshop on Sketch-Based Interfaces and Modeling, The Eurographics Association, 2012.

[17] S. Pirk, O. Stava, J. Kratt, M. A. M. Said, B. Neubert, R. Měch, B. Benes, O. Deussen, Plastic trees: Interactive self-adapting botanical tree models, ACM Trans. Graph. 31 (4) (2012) 50:1–50:10.

[18] S. Pirk, T. Niese, T. Hädrich, B. Benes, O. Deussen, Windy trees: Computing stress response for developmental tree models, ACM Trans. Graph. 33 (6) (2014) 204:1–204:11.

[19] I. Shlyakhter, M. Rozenoer, J. Dorsey, S. Teller, Reconstructing 3D Tree Models from Instrumented Photographs, IEEE Computer Graphics and Applications 21 (3) (2001) 53–61.

[20] A. Reche, I. Martin, G. Drettakis, Volumetric Reconstruction and Interactive Rendering of Trees from Photographs, ACM Transactions on Graphics 23 (3) (2004) 720–727.

[21] B. Neubert, T. Franken, O. Deussen, Approximate Image-based Tree-modeling Using Particle Flows, ACM Trans. Graph. 26 (3).

[22] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, S. B. Kang, Image-based Plant Modeling, ACM Trans. Graph. 25 (3) (2006) 599–604.

[23] P. Tan, G. Zeng, J. Wang, S. B. Kang, L. Quan, Image-based Tree Modeling, ACM Transactions on Graphics 26 (3) (2007) 87.

[24] D. Bradley, D. Nowrouzezahrai, P. Beardsley, Image-based Reconstruction and Synthesis of Dense Foliage, ACM Trans. Graph. 32 (4) (2013) 74:1–74:10.

[25] H. Xu, N. Gossett, B. Chen, Knowledge and heuristic-based modeling of laser-scanned trees, ACM Trans. Graph. 26 (4).

[26] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, J. El-Sana, Automatic Reconstruction of Tree Skeletal Structures from Point Clouds, ACM Transactions on Graphics 29 (6) (2010) 151:1–151:8.

[27] Y. Livny, S. Pirk, Z. Cheng, F. Yan, O. Deussen, D. Cohen-Or, B. Chen, Texture-lobes for Tree Modelling, ACM Trans. Graph. 30 (4) (2011) 53:1–53:10.

[28] M. Okabe, S. Owada, T. Igarashi, Interactive Design of Botanical Trees using Freehand Sketches and Example-based Editing, Computer Graphics Forum 24 (3) (2005) 487–496.

[29] J. Wither, F. Boudon, M.-P. Cani, C. Godin, Structure from silhouettes: a new paradigm for fast sketch-based design of trees, Computer Graphics Forum 28 (2) (2009) 541–550.

[30] P. Tan, T. Fang, J. Xiao, P. Zhao, L. Quan, Single Image Tree Modeling, ACM Transactions on Graphics 27 (5) (2008) 108:1–108:7.

[31] F. Boudon, A. Meyer, C. Godin, Survey on Computer Representations of Trees for Realistic and Efficient Rendering, Rapport de recherche 2301, LIRIS, Université Claude Bernard Lyon 1 (2006).

[32] W. T. Reeves, R. Blau, Approximate and probabilistic algorithms for shading and rendering structured particle systems, SIGGRAPH Comput. Graph. 19 (3) (1985) 313–322.

[33] O. Deussen, C. Colditz, M. Stamminger, G. Drettakis, Interactive visualization of complex plant ecosystems, in: Proceedings of the Conference on Visualization '02, VIS '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 219–226.

[34] G. Gilet, A. Meyer, F. Neyret, Point-based rendering of trees, in: Proceedings of the First Eurographics Conference on Natural Phenomena, 2005.

[35] P. Decaudin, F. Neyret, Rendering forest scenes in real-time., in: A. Keller, H. W. Jensen (Eds.), Rendering Techniques, Eurographics Association, 2004, pp. 93–102.

[36] C. Andújar, A. Chica, M. A. Vico, S. Moya, P. Brunet, Inexpensive reconstruction and rendering of realistic roadside landscapes, Computer Graph-

ics Forum 33 (6) (2014) 101–117.
[37] R. L. Cook, J. Halstead, M. Planck, D. Ryu, Stochastic simplification of aggregate detail, in: ACM SIGGRAPH 2007 Papers, SIGGRAPH '07, ACM, New York, NY, USA, 2007.
[38] B. Neubert, S. Pirk, O. Deussen, C. Dachsbacher, Improved model- and view-dependent pruning of large botanical scenes, Computer Graphics Forum 30 (6) (2011) 1708–1718.
[39] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, O. Deussen, Realistic real-time rendering of landscapes using billboard clouds, Computer Graphics Forum (Proceedings of EUROGRAPHICS 2005), 24(3).
[40] E. Bruneton, F. Neyret, Real-time Realistic Rendering and Lighting of Forests, Computer Graphics Forum 31 (2pt1) (2012) 373–382.
[41] G. Friedland, K. Jantz, R. Rojas, Siox: simple interactive object extraction in still images, in: Multimedia, Seventh IEEE International Symposium on, 2005, pp. 7 pp.–.
[42] T. Igarashi, S. Matsuoka, H. Tanaka, Teddy: a sketching interface for 3d freeform design, in: ACM SIGGRAPH 1999, ACM, 1999, pp. 409–416.
[43] L. Kobbelt, S. Campagna, J. Vorsatz, H.-P. Seidel, Interactive multi-resolution modeling on arbitrary meshes, in: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '98, 1998, pp. 105–114.
[44] P. Harrison, Image texture tools, Ph.D. thesis, PhD thesis, Monash University (2005).
[45] M. Glencross, G. J. Ward, F. Melendez, C. Jay, J. Liu, R. Hubbold, A perceptually validated model for surface depth hallucination, in: ACM SIGGRAPH 2008 Papers, ACM, 2008, pp. 59:1–59:8.
[46] P. J. Burt, E. H. Adelson, The laplacian pyramid as a compact image code, IEEE Transactions on Communications 31 (1983) 532–540.
[47] F. Policarpo, M. M. Oliveira, J. L. D. Comba, Real-time relief mapping on arbitrary polygonal surfaces, in: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games, I3D '05, ACM, New York, NY, USA, 2005, pp. 155–162.
[48] A. Beacco, N. Pelechano, C. Andujar, A survey of real-time crowd rendering, Computer Graphics Forum Article first published online: 15 Oct 2015. doi:10.1111/cgf.12774.
[49] L. Ruiz, A. Fdez-Sarría, J. Recio, Texture feature extraction for classification of remote sensing data using wavelet decomposition: a comparative study, in: 20th ISPRS Congress, Vol. 35, 2004, pp. 1109–1114.